浙江大学
ZHEJIANG UNIVERSITY

Computer Architecture Experiment

Topic 1. Single-cycle CPU Design

浙江大学计算机学院
陈文智
chenwz@zju.edu.cn

实验操作流程

- 阅读实验文档，理解单周期处理器的各个功能模块组成和内部实现方式。
- 补全各个功能模块源代码中的空缺部分。
- 对处理器进行仿真，检验处理器的仿真结果是否符合要求。
- 综合工程并下载至开发板，在单步执行的过程中检查调试屏幕的输出，检验处理器的执行过程是否正确。

实验验收标准

- 仿真执行过程中，处理器的行为和内部控制信号均符合要求。
- 下载至开发板后的单步执行过程中，寄存器的变化过程和最终执行结果与测试程序相吻合。

Outline

- Experiment Purpose
- Experiment Task
- Basic Principle
- Operating Procedures
- Precaution
- Checkpoints

Experiment Purpose

- **Understand the principles of single-cycle CPU controller and master methods of single-cycle CPU controller design**
- **Understand the principles of datapath and master methods of datapath design**
- **Understand the principles of single-cycle CPU and master methods of single-cycle CPU design**
- **master methods of program verification of CPU**

Experiment Task

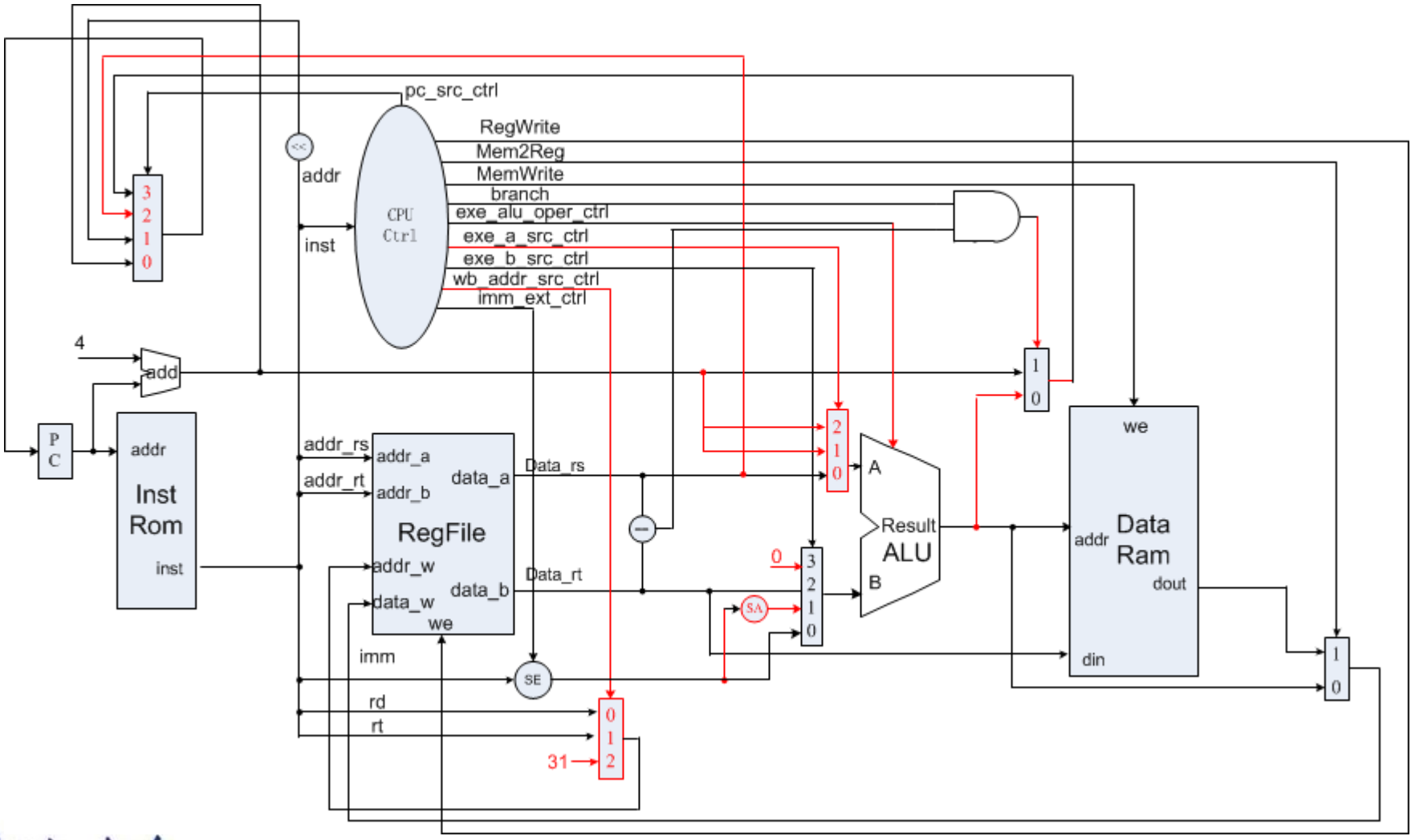
- **Design the CPU Controller, Datapath, bring together the basic units into Single-cycle CPU**

- **Verify the Single-Cycle CPU with program and observe the execution of program**

16 MIPS Instructions

Bit #	31..26	25..21	20..16	15..11	10..6	5..0	Operations	
R-type	op	rs	rt	rd	sa	func		
add	000000	rs	rt	rd	00000	100000	rd = rs + rt; with overflow	PC+=4
sub		rs	rt	rd	00000	100010	rd = rs - rt; with overflow	PC+=4
and		rs	rt	rd	00000	100100	rd = rs & rt;	PC+=4
or		rs	rt	rd	00000	100101	rd = rs | rt;	PC+=4
sll		00000	rt	rd	sa	000000	rd = rt << sa;	PC+=4
srl		00000	rt	rd	sa	000010	rd = rt >> sa (logical);	PC+=4
slt		rs	rt	rd	00000	101010	if(rs < rt)rd = 1; else rd = 0; <(signed)	PC+=4
jr		rs	00000	00000	00000	001000		PC=rs
I-type	op	rs	rt	immediate				
addi	001000	rs	rt	imm			rt = rs + (sign_extend)imm; with overflow	PC+=4
andi	001100	rs	rt	imm			rt = rs & (zero_extend)imm;	PC+=4
ori	001101	rs	rt	imm			rt = rs | (zero_extend)imm;	PC+=4
lw	100011	rs	rt	imm			rt = memory[rs + (sign_extend)imm];	PC+=4
sw	101011	rs	rt	imm			memory[rs + (sign_extend)imm] <-- rt;	PC+=4
beq	000100	rs	rt	imm			if (rs == rt) PC+=4 + (sign_extend)imm <<2;	PC+=4
J-type	op	address						
j	000010	address					PC = (PC+4)[31..28],address<<2	
jal	000011	address					PC = (PC+4)[31..28],address<<2 ; $31 = PC+4	

CPU Controller



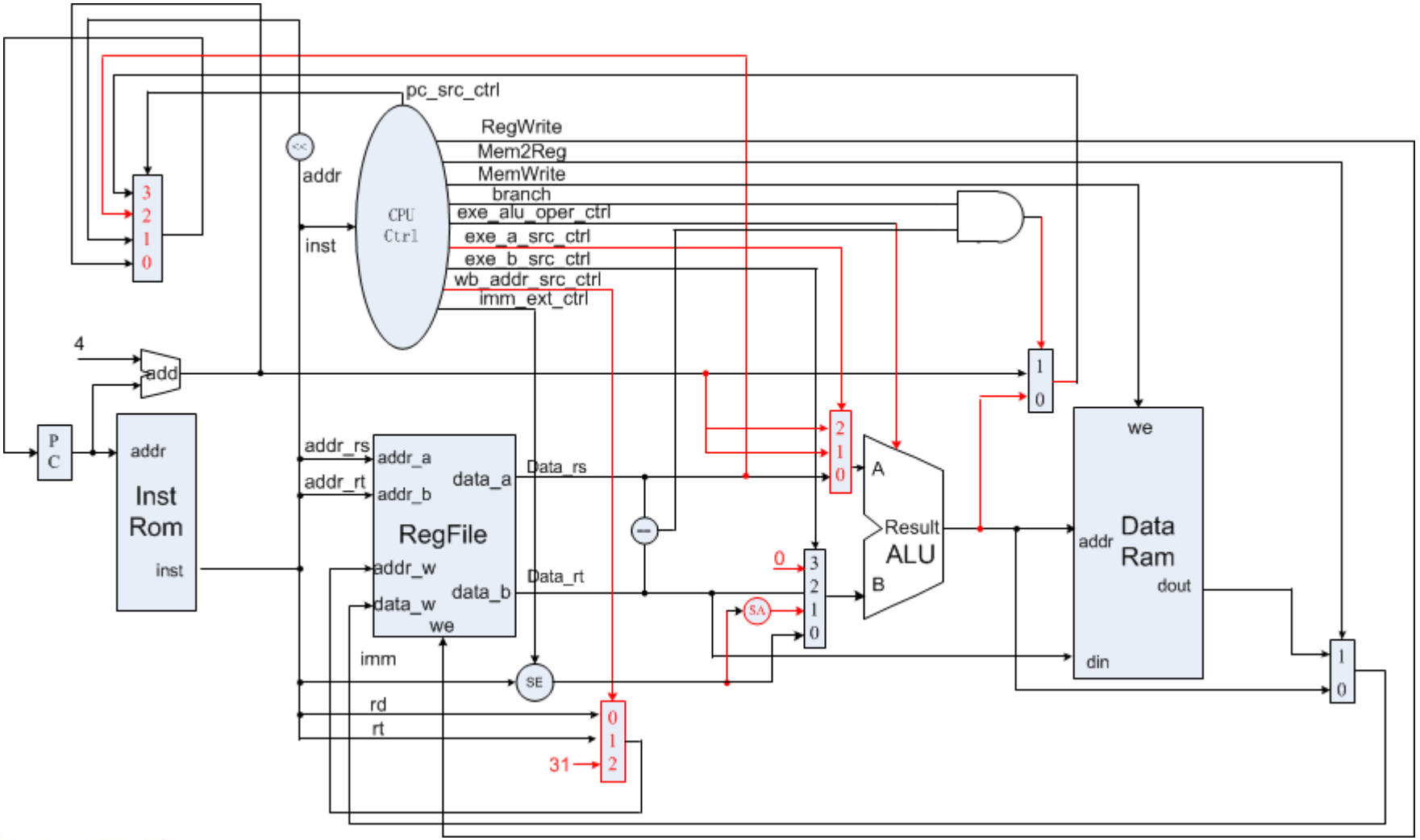
Output of CPU Controller(1)

- pc_src_ctrl(inst_addr)
  - Default: PC+4
  - PC_JUMP: {inst_addr[31:28], inst_data[25:0], 2'b0}
  - PC_JR: data_rs
  - PC_BEQ: alu_out
- wb_addr_src_ctrl (regw_addr)
  - WB_ADDR_RD: addr_rd
  - WB_ADDR_RT: addr_rt
  - WB_ADDR_LINK: GPR_RA

Output of CPU Controller(2)

- exe_a_src_ctrl(opa)
  - EXE_A_RS: data_rs
  - EXE_A_LINK: inst_addr_next
  - EXE_A_BRANCH: inst_addr_next
- exe_b_src_ctrl(opb)
  - EXE_B_RT: data_rt
  - EXE_B_IMM: data_imm
  - EXE_B_LINK: 32'h0
  - EXE_B_BRANCH: {data_imm[29:0], 2'b0}

Datapath


pc_src_ctrl
RegWrite
Mem2Reg
MemWrite
branch
exe_alu_oper_ctrl
exe_a_src_ctrl
exe_b_src_ctrl
wb_addr_src_ctrl
imm_ext_ctrl
CPU Ctrl
addr
inst
<<
4
add
PC
addr
Inst Rom
inst
addr_rs
addr_rt
addr_a
addr_b
data_a
data_b
RegFile
addr_w
data_w
we
Data_rs
Data_rt
imm
SE
rd
rt
31
A
B
Result ALU
SA
0
we
addr
Data Ram
dout
din

Basic Units of Single-cycle CPU

- **CPU Controller**
- **ALU**
- **Register file**
- **Instruction Mem. and Data Mem.**
- **others： Register, sign-extend Unit, shifter, multiplexer**

Instruction Memory

```
parameter
        ADDR_WIDTH = 6;

reg [31:0] data [0:(1<<ADDR_WIDTH)-1];

initial   begin
        $readmemh("inst_mem.hex", data);
end

always @(*) begin
        if (addr[31:ADDR_WIDTH] != 0)
                dout = 32'h0;
        else
                dout = data[addr[ADDR_WIDTH-1:0]];
end
```

Data Memory(1)

```
parameter
        ADDR_WIDTH = 5;

reg [31:0] data [0:(1<<ADDR_WIDTH)-1];

initial   begin
        $readmemh("data_mem.hex", data);
end

always @(negedge clk) begin
        if (we && addr[31:ADDR_WIDTH]==0)
                data[addr[ADDR_WIDTH-1:0]] <= din;
end
```

Data Memory(2)

```
reg [31:0] out;
always @(negedge clk) begin
        out <= data[addr[ADDR_WIDTH-1:0]];
end

always @(*) begin
        if (addr[31:ADDR_WIDTH] != 0)
                dout = 32'h0;
        else
                dout = out;
end
```

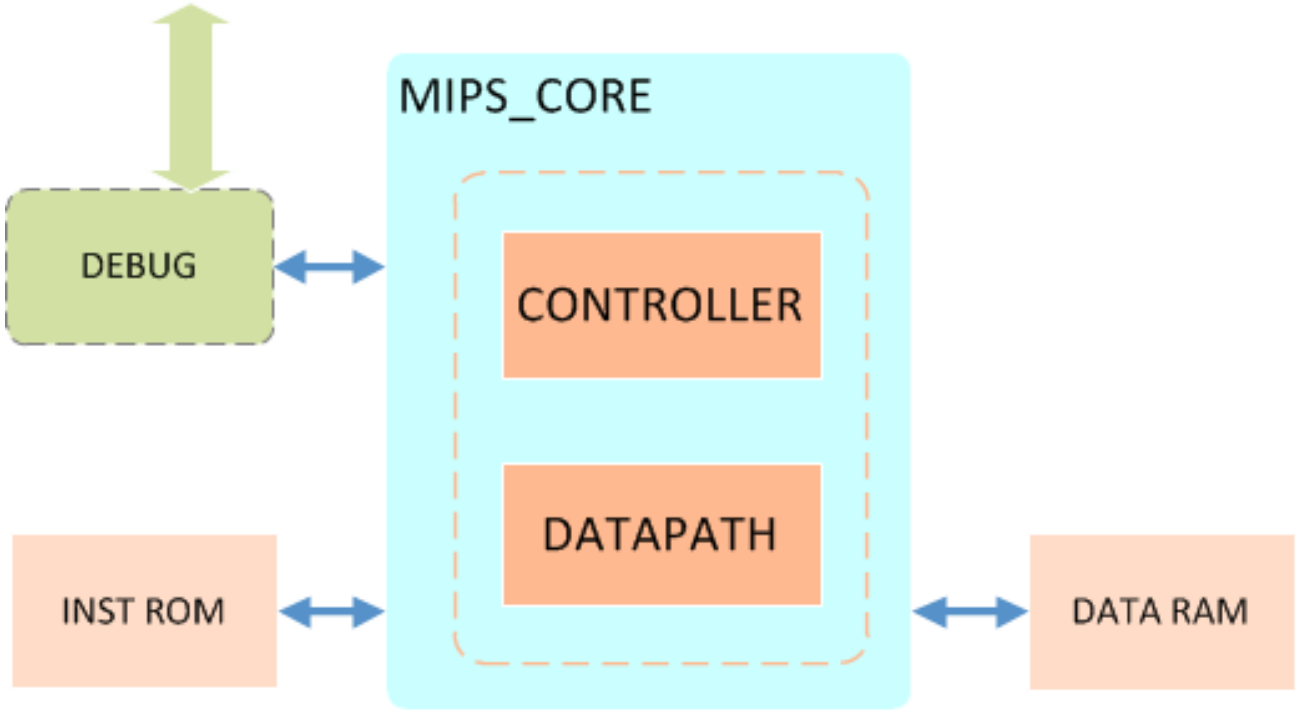
Register File

```
reg [31:0] regfile [1:31];  // $zero is always zero
// write
always @(posedge clk) begin
        if (en_w && addr_w != 0)
                regfile[addr_w] <= data_w;
end
// read
always @(*) begin
        data_a = addr_a == 0 ? 0 : regfile[addr_a];
        data_b = addr_b == 0 ? 0 : regfile[addr_b];
end
// debug
`ifdef DEBUG
always @(*) begin
        debug_data = debug_addr == 0 ? 0 : regfile[debug_addr];
end
`endif
```

Timing for single-cycle

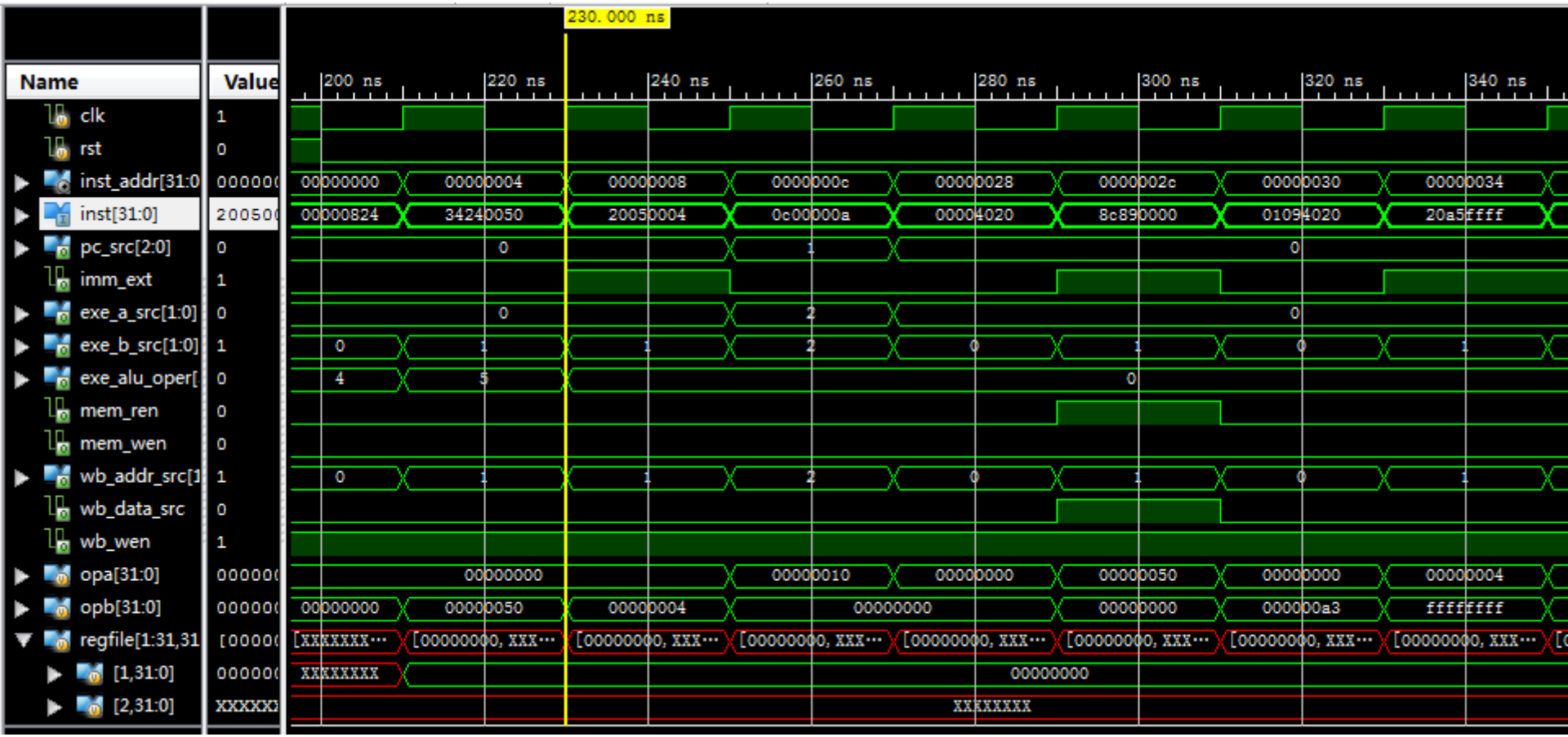
- Instruction Mem. Read: anytime
- Regfile Read: anytime
- Data Mem. Read/Write: Negtive Edge
- Regfile Write: Positive Edge

SC CPU Diagram

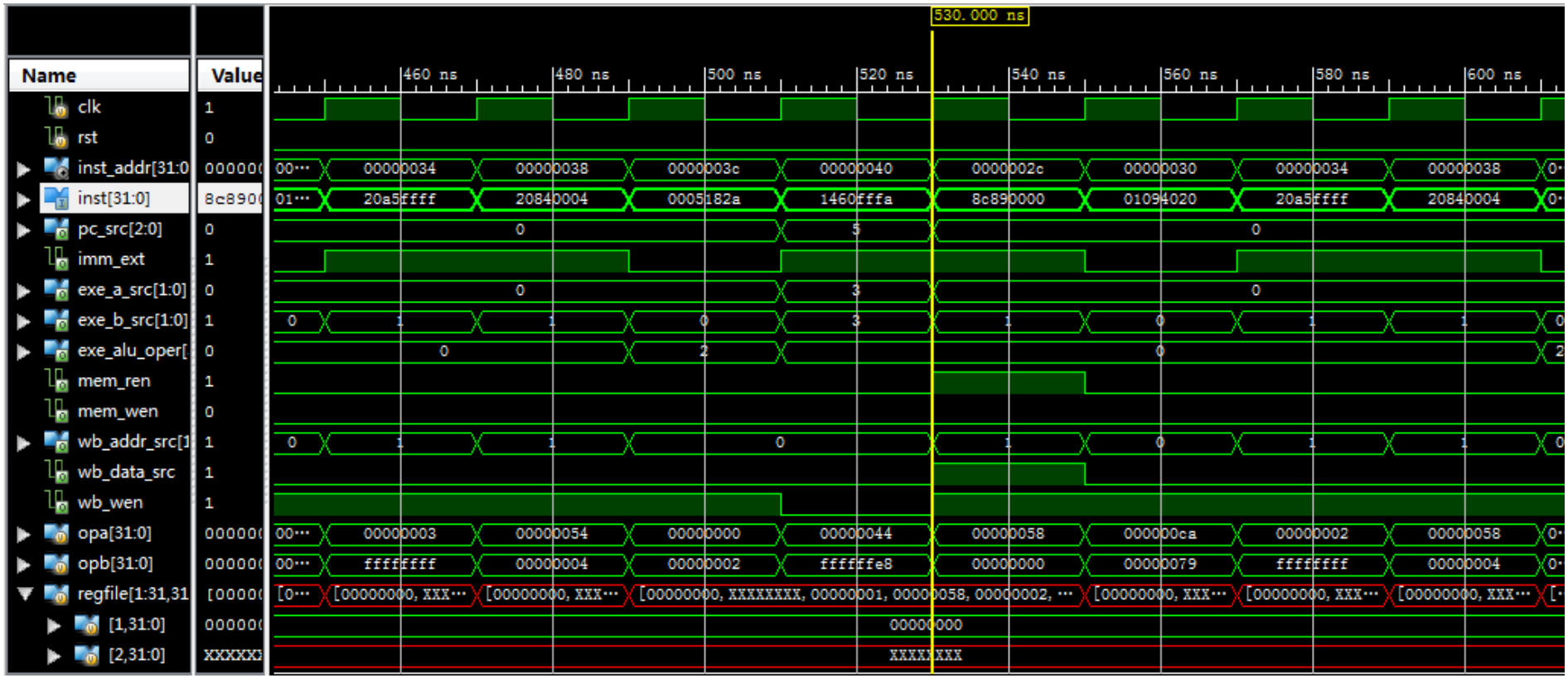


Simulation (1)

Simulation (2)



Simulation (3)



Program for verification (1)

```
00000824 main:    and $1, $0, $0      # address of data[0]
34240050          ori $4, $1, 80      # address of data[0]
20050004 call:    addi $5, $0, 4      # counter
0c00000a          jal sum             # call function
ac820000 return:  sw $2, 0($4)        # store result
8c890000          lw $9, 0($4)        # check sw
ac890004          sw $9, 4($4)        # store result again
01244022          sub $8, $9, $4      # sub: $8 <- $9 - $4
08000008 finish:  j finish            # dead loop
00000000          nop                 # done
```

Program for verification (2)

```
00004020 sum:   add $8, $0, $0      # sum function entry
8c890000 loop:  lw $9, 0($4)        # load data
01094020        add $8, $8, $9      # sum
20a5ffff        addi $5, $5, -1     # counter - 1
20840004        addi $4, $4, 4      # address + 4
0005182a        slt $3, $0, $5      # finish?
1460fffa        bne $3, $0, loop    # finish?
01001025        or $2, $8, $0       # move result to $v0
03e00008        jr $ra              # return
00000000        nop                 # done
```

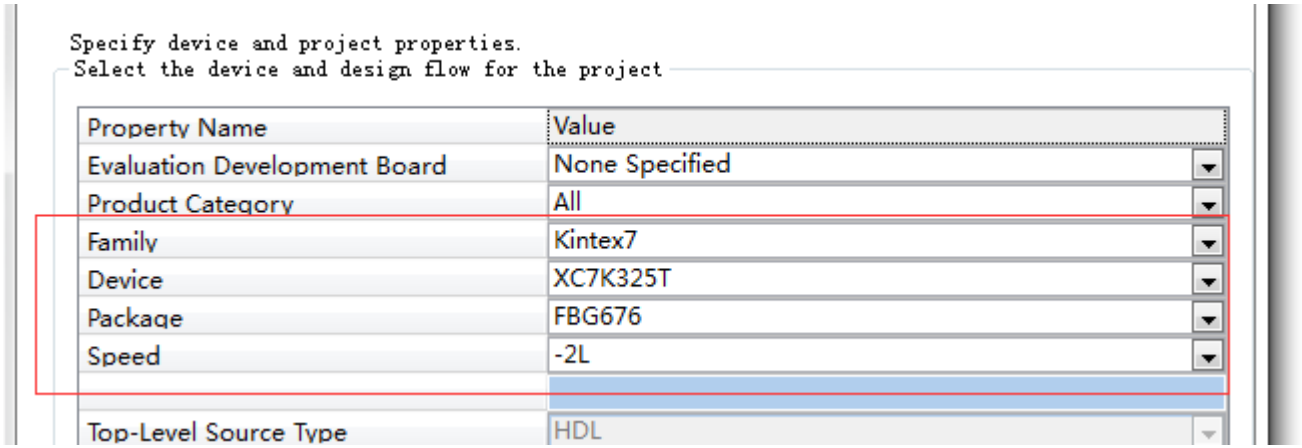
Checkpoints

- **CP 1: Waveform Simulation of Single-cycle CPU**

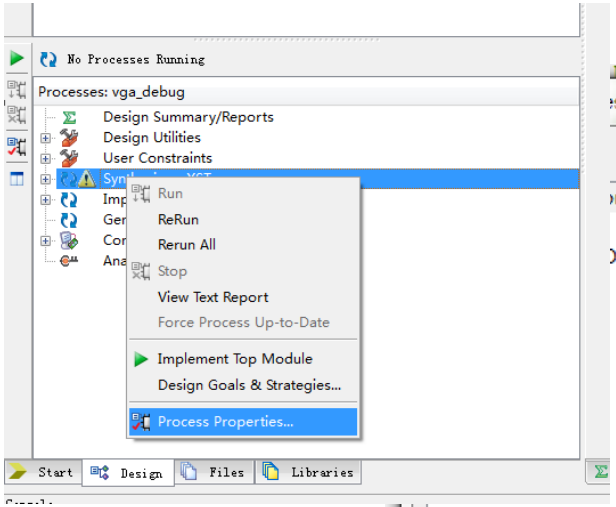
- **CP 2: FPGA Implementation of Single-cycle CPU with the verification program**

FPGA Implementation

- Project Settings
  - Top Module: mips_top



FPGA Implementation



Module Name: vga_debug

s - Synthesis Options

Properties are not editable while a process is running.

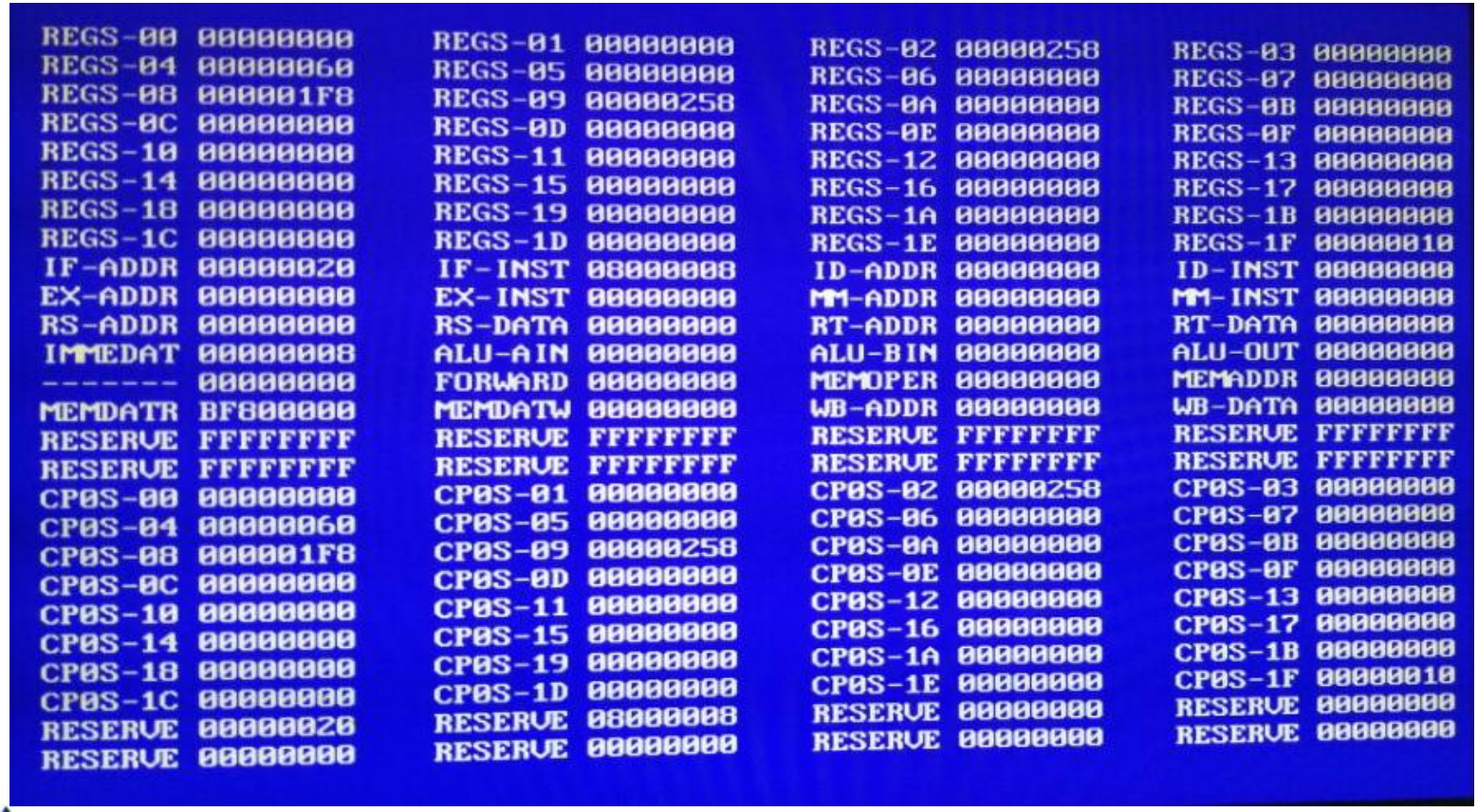
Switch Name	Property Name	Value
-opt_mode	Optimization Goal	Speed
-opt_level	Optimization Effort	Normal
-power	Power Reduction	
-iuc	Use Synthesis Constraints File	☑
-uc	Synthesis Constraints File	
-keep_hierarchy	Keep Hierarchy	Soft
-netlist_hierarchy	Netlist Hierarchy	As Optimized
-glob_opt	Global Optimization Goal	AllClockNets
-rtlview	Generate RTL Schematic	Yes
-read_cores	Read Cores	☑
-sd	Cores Search Directories	
-write_timing_constraints	Write Timing Constraints	
-cross_clock_analysis	Cross Clock Analysis	
-hierarchy_separator	Hierarchy Separator	/
-bus_delimiter	Bus Delimiter	<>
-slice_utilization_ratio	LUT-FF Pairs Utilization Ratio	100
-bram_utilization_ratio	BRAM Utilization Ratio	100
-dsp_utilization_ratio	DSP Utilization Ratio	100
-case	Case	Maintain
	Work Directory	D:\Homework\ARC\2017\student_test\xst
set -xsthdpini	HDL INI File	
	Library for Verilog Sources	
-lso	Library Search Order	
-vlgincdir	Verilog Include Directories	
-generics	Generics, Parameters	
-define	Verilog Macros	
	Other XST Command Line Options	

Property display level: Advanced ☑ Display switch names Default

浙江大学 ZheJiang University

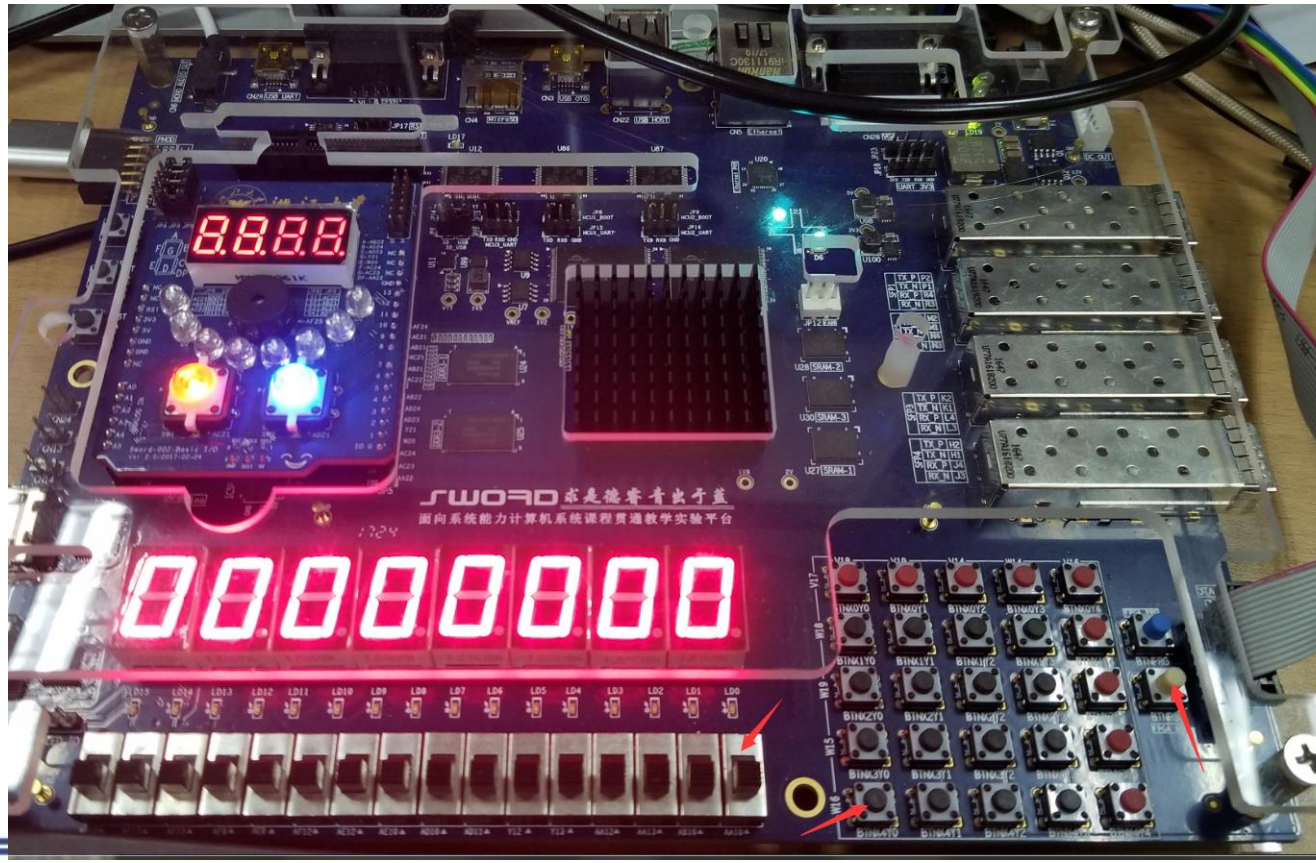
FPGA Implementation

- VGA-based debugger



FPGA Implementation

- Control
  - Enable Single Step: SW[0]
  - Step: BTNX4Y0
  - Reset: BTNRST



Thanks!

浙江大学
ZheJiang University